WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Tuesday, November 16, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR	
☐	L25	L24 and (single adj3 (ROM or RAM or EPROM or EEPROM or memory))	29
☐	L24	L23 and (multiple adj3 (processing or processor or CPU))	93
☐	L23	19991125	471
☐	L22	L20 and l14	0
☐	L21	L20 and l15	0
☐	L20	709/213.ccls.	836
☐	L19	l15 and (multiple adj3 (processing or processor or CPU))	8
☐	L18	l17 and (multiple adj3 (processing or processor or CPU))	0
☐	L17	L16 and (initialize or initialization)	21
☐	L16	L15 and register	56
☐	L15	19991125	70
☐	L14	(CPU or processor)near8 (access or accessing) near8 (memory or EPROM or EEPROM) near8 (notify or notification)	161
☐	L13	CPU near8 (access or accessing) near8 (memory or EPROM or EEPROM) near8 (notify or notification)	87
☐	L12	L11 and l10	22
☐	L11	L8 and (single adj3 (ROM or RAM or EPROM or EEPROM or memory))	223
☐	L10	L8 and (multiple adj3 (processing or processor or CPU))	263
☐	L9	L8 and (multiple adj3 (processing or CPU))	125
☐	L8	19991125	46805
☐	L7	l3 and (engine or vehicle)	76483
☐	L6	l2 and (engine or vehicle)	1
☐	L5	l3 and l1	0
☐	L4	L3 and l2	0
☐	L3	electronic near5 control	227791
☐	L2	19991125)	23
☐	L1	((multiple adj3 (processing or CPU)) near8(single adj3 (ROM or RAM or EPROM or EEPROM or memory)))	34

END OF SEARCH HISTORY

L7: Entry 11 of 17 File: USPT Aug 24, 1993

DOCUMENT-IDENTIFIER: US 5239634 A
TITLE: Memory controller for enqueuing/dequeuing process

Application Filing Date (1):
19890921

Brief Summary Text (5):
CPU's perform queue manipulations by executing a sequence of read and write operations which are issued to the memory system. Some CPU's include such manipulations in their instruction repertoire, while others require explicit programming of the individual steps involved. In either case, the data which is read and written during the queue manipulation passes into and out of the CPU. In systems with multiple CPU's served by a single memory controller, the CPU's must take explicit actions to assure that queue manipulations are "atomic" sequences. "Atomic" sequences provide exclusive access to the memory controller when one of the CPU's begins a queue manipulation until that CPU finishes the queue manipulation. Otherwise, the sequence of operations necessary to perform a proper queue modification by one CPU would be corrupted by the operations of another CPU simultaneously modifying the same queue. This can leave the queue in an inconsistent state.

Detailed Description Text (2):
Referring to the drawings, wherein reference characters designate like or corresponding parts throughout the views, FIG. 1 shows the arrangement of a typical data processing system in which the present invention is used. A data processing system 2 typically comprises a plurality of CPU's 4. Although four of the CPU's 4 are shown in FIG. 1, the system 2 may have a greater or lesser number of the CPU's 4, but at least one of the CPU's 4. The CPU's 4 are coupled to a shared memory system 6 via a common data bus 8.

L7: Entry 12 of 17 File: USPT Sep 8, 1992

DOCUMENT-IDENTIFIER: US 5146607 A
TITLE: Method and apparatus for sharing information between a plurality of processing units

Abstract Text (1):
A plurality of processing units, each having a local memory connected thereto is disclosed. A write sense controller is also connected to each of the processing units to transmit a memory write word into a shared portion of local memory over a reflective memory line. Other write sense controllers receive the memory words from the reflective memory bus and cause them to be written into corresponding storage locations in the shared partitions of their local memories.

Application Filing Date (1):
19910909

Brief Summary Text (2):
The field of the invention relates to multiple processor digital computer systems. More particularly the field of the invention relates to multiple processor digital computer systems having means for sharing substantially identical blocks of information stored within a plurality of local main memory units.

Brief Summary Text (6):
It has also been proposed that a shared memory be subdivided into local memories. However, in that case, it has been impossible to maintain the integrity of the local memories over the wide range of addresses which might be accessed by a particular processor.

Brief Summary Text (7):
What is needed then is a method and apparatus which can allow multiple processors to execute simultaneously various portions of code while rapidly and efficiently sharing information between themselves.

Brief Summary Text (9):
A multiple processor shared memory system is disclosed herein. A plurality of central processor units has connected thereto respective local buses which are adapted to carry data, address, timing and control signals thereon. A dual port primary storage or memory unit having a first port and a second port has its first port connected to the local bus for exchange of information therewith.

Brief Summary Text (12):
A plurality of write sense controllers together with associated central processor units, memories and read sensor controllers may be connected to the reflective memory bus so that each local memory has a first addressing range identified as a shared memory portion and a second addressing range identified as a local memory portion. The shared memory portions have identical contents.

Brief Summary Text (13):
It is a principal object of the present invention to provide a multiple processor data processing system wherein each processor has a single local memory which

stores a shared memory segment, identical to shared memory segments in local memory units of other processors.

Brief Summary Text (14):
It is an additional object of the instant invention to provide a multiple processor unit data processing system having a shared memory structure with minimum contention and latency delays.

Detailed Description Text (7):
As may best be seen in FIG. 2, the memory units are segmented into shared areas and local areas. In the present embodiment, the memory unit 32 has a shared area 60 and a local area 62. The memory unit 34 has a shared area 64 and a local area 66. The memory unit 36 has a shared area 68 and a local area 70. The shared areas reside in a plurality of memory cells between a first set of addresses in a particular memory unit. The local area resides in a plurality of memory cells having a second address range in a particular memory. Thus, the selection of a particular address within a memory selects whether data is local data or shared data.

Detailed Description Text (10):
The write sense controller 38 tests whether the memory write operation is to a memory location within either the shared region or the local region. In the event that the address carried on the local bus 26 is within the range of addresses occupied by the shared region, the write sense controller retransmits the data to be written into the shared region of memory unit 32 on the link 44 and also onto the reflective memory bus 12 where that information is received by the write sense controllers 40 and 42 as will be seen in more detail hereafter.

Detailed Description Text (14):
At the same time, the address is fed directly to a lower address comparator 94 and an upper address comparator 96 by an address input bus 100 connected between the comparators 94 and 96 and the bus 84. In the event that the address so supplied to the lower address comparator 94 is not less than the address fed from the control register latch 1 for a lower address limit and not greater than the address fed to the upper address comparator 96, respective true outputs are provided at a pair of leads 102 and 104 which are respectively connected from the lower address comparator 94 and the upper address comparator 96 to the write decode logic 86. The two signals are used to enable the write decode logic 86 which in turn is connected via a lead 106 to the second register latch 82 to enable the second register latch 82 to receive data from the first register latch 80. In other words, the register latch 82 will only be enabled to receive data when the address to which the data is written is within the address bounds stored in the control register latch 1. Those address bounds define the shared area of the local memory 32. The address boundaries by the setting of the address decode jumpers 92 which load the lower and upper addresses for write sense comparator transmission to the reflective memory bus into the first control register 88.

Detailed Description Text (15):
Assuming that the data is to be written into the shared area of the memory, it is then received by the latch 82 and outputted via a bus 108 connected to the latch 82 to an offset subtract logic module 110. The offset subtract logic module subtracts any address offset from the address portion of the memory write word so that a normalized address can be supplied to the reflective memory bus.

Detailed Description Text (16):
The data is then transmitted further by the bus 108 to a parity generator 112 which generates by the parity signals on the transmitted address and data. The data, normalized addressed, control and timing signals are then supplied by the bus 108 to a plurality of request fifo registers 114. Each time a word is transferred to the request fifos 114 from the bus 108, a request counter 116, which is connected by a lead 118, is incremented. In the event that the request counter 116 exceeds a

preset count, in the preferred embodiment 56, a signal is supplied to the connector 118 and to certain inhibit and priority jumpers 120 to cause an inhibit signal to be placed on the local bus 26 to prevent further memory write operations from occurring on the local bus 26 in the shared memory range. When the request counter 116 indicates that there are less than 56 words present in the request fifos 114 the local bus 26 is enabled for memory write operation. The words are then transmitted via a bus 122 from the request fifos 114 to a plurality of reflective memory bus transceivers 124. Each time that a word is transmitted from the fifos, the request counter logic 116 is decremented. The words are then transmitted via the bus transceivers 124 to the reflective memory bus 12 for receipt by the other write sense controllers 40 and 42.

Detailed Description Text (19):
Thus when the central processor unit 20 performs a local memory write operation into its own memory 32 at its own shared address range, the same memory write is passed through the write sense controller 38 with any address offsetting which must be performed and supplied via the reflective memory bus 12 to all other write sense controllers connected thereto. Therefore, each time a memory location within the shared region of a local memory is updated, the updating information as well as its address is broadcast over the reflective memory bus 12.

Detailed Description Text (21):
The information word taken from the reflective memory bus 12 is then supplied via a bus 152 to an address offset addition logic module 154 which is connected thereto. Any offsetting address which is necessary is then added to the received address and the address information is fed via a bus 156 to an upper link address comparator 158 and a lower link address comparator 160 for the purpose of determining whether the local shared memory partition encompasses the address so supplied by the addition logic. The local partition is defined by information supplied from the address decode jumpers 92 to the second control register latch 90 which is connected via a bus 162 to the upper link address comparators 158 and the lower link address comparator 160. In most instances, the shared partition will have identical upper and lower address bounds whether the write sense controller is operating in a transmitting or a receive mode. There are instances, however, when different bounds may be desirable to be used. Thus, separate comparator circuitry has been provided to detect when the information received from the reflective memory bus is within the receive shared partition.

Detailed Description Text (22):
In the event that the information is within the address limits of the shared partition, enabling signals are supplied to a pair of leads 164 and 166 which are respectively connected to the upper link address comparator 158 and the lower link address comparator 160 and to a write sense controller-read sense controller bus control logic module 168. That module 168 is enabled to cause the information supplied to a bus 170 by the reflective memory bus receive latch 150 and to the write sense controller and read sense controller interface bus 172 connected thereto to be latched into the read sense controller 50 after having been cleared by a subparity checker 174.

Detailed Description Text (23):
In summary, when a signal from the write sense controller 40 or the write sense controller 42 is supplied to the reflective memory bus 12, the write sense controller 38 has its reflective memory bus receive latch 150 enabled to latch the information therein. The received address is then offset and compared to the delimiting addresses of the shared region. In the event that the received address, as offset, is within the shared region, the information from the latch which is already present on the write sense controller-read sense controller interface bus 172 is latched into the input of the read sense controller 50.

Detailed Description Text (45):

It may be appreciated then, that the multiprocessing system provides a high speed method of sharing information between a plurality of processors. When information within a particular shared region of a local memory is updated in the local memory, it is simultaneously transmitted by a local write sense controller onto the reflective memory bus received by all other write sense controllers and stored at the corresponding shared addresses in their local memories by the read sense controllers. When it is necessary for a local processor to perform a memory access operation to read data from a shared portion of the data structure, a local memory is read from. All other local memories continue operating independently allowing all other local processors to continue on with their operations and completely eliminating contention and latency delays when information is read from the shared portion of a particular memory.

CLAIMS:

1. A data processing system, comprising:

a data bus; and

a plurality of nodes connected to the data bus, at least one of said plurality of nodes comprises:

processor means for generating a data signal, an address signal and a memory write signal;

a dual port memory unit having a first storage area for storing data local to a respective node and a second storage area for storing data shared between the plurality of nodes;

a local bus connecting the processor means to a first port of the memory unit; and

write sensing means connected to the data bus and also to the processing unit through the local bus, said write sensing means comprising:

first comparator means for comparing address signals sensed from the local bus with a first predetermined value;

second comparator means for comparing the address signals sensed from the local bus with a second predetermined value;

first latch means for holding the data signals, address signals and write signals;

first offset logic means for normalizing the address signal in response to the address signal being released from the first latch means;

first request FIFO means for receiving data signals, and write signals from the first latch means, in response to receipt of a signal by the first latch means indicating that the address signal is of a value larger the first predetermined value, of a value smaller than the second predetermined value, and address signals from the first offset logic means;

transmitter means for transmitting the sensed data signals, normalized address signals and memory write signals in order of receipt by the first request FIFO means;

third comparator means for comparing address signals sensed from the data bus with a third predetermined value;

fourth comparator means for comparing the address signals sensed from the data bus with a fourth predetermined value; and

second offset logic means for offsetting the address signal to its original value upon a determination that the address signal is of a value larger than the third predetermined value and of a value smaller than the fourth predetermined value;

the at least one of the processing nodes further comprising read sensing means connected between the write sensing means and a second port of the memory, unit, said read sensing means comprising:

correction logic means for generating corrected signals for data sensed from the data bus by the write sensing means;

second request FIFO means for receiving data signals, address signals and write sense signals from the write sense logic; and

combining means for combining the correction signals with the data signals producing a corrected data signal to be delivered to the shared storage area of the memory unit.

2. A processing system as claimed in claim 1, wherein the read sensing means further comprises:

alignment decoder means for determining the length of the data signal; and

byte alignment means for adapting data of differing lengths for placement into the shared storage area of the memory unit.

4. A data processing system, comprising:

a data bus; and

a plurality of nodes connected to the data bus, at least one of said plurality of nodes comprises:

processor means for generating a data signal, an address signal and a memory write signal;

a dual port memory unit having a first storage area for storing data local to a respective node and a second storage area for storing data shared between the plurality of nodes;

a local bus connecting the processor means to a first port of the memory unit; and

write sensing means connected to the data bus and also to the processing unit through the local bus, said write sensing means comprising:

first comparator means for comparing address signals sensed from the local bus with a first predetermined value;

second comparator means for comparing the address signals sensed from the local bus with a second predetermined value;

first latch means for holding the data signals, address signals and write signals;

first offset logic means for normalizing the address signal in response to the address signal being released from the first latch means;

first request FIFO means for receiving data signals, and write signal from the first latch means, in response to receipt of a signal by the first latch means indicating that the address signal is of a value larger the first predetermined

value, of a value smaller than the second predetermined value, and address signals from the first offset logic means;

transmitter means for transmitting the sensed data signals, normalized address signals and memory write signals in order of receipt by the first request FIFO means;

third comparator means for comparing address signals sensed from the data bus with a third predetermined value;

fourth comparator means for comparing the address signals sensed from the data bus with a fourth predetermined value; and

second offset logic means for offsetting the address signal to its original value upon a determination that the address signal is of a value larger than the third predetermined value. and of a value smaller than the fourth predetermined value.

5. A data processing system, as claimed in claim 4 wherein at least one of said plurality of nodes comprises:

processor means for generating a data signal, an address signal and a memory write signal;

a dual port memory unit having a first storage area for storing data local to a respective node and a second storage area for storing data shared between the plurality of nodes;

a local bus connecting the processor means to a first port of the memory unit; and

read sensing means connected between the write sensing means and a second port of the memory unit, said read sensing means comprising:

correction logic means for generating corrected signals for data sensed from the data bus by the write sense means;

second request FIFO means for receiving data signals, address signals and write sense signals from the write sense logic; and

combining means for combining the correction signals with the data signals producing a corrected data signal to be delivered to the shared storage area of the memory unit.

6. Processing system as claimed in claim 5, wherein the read sensing means further comprises:

alignment decoder means for determining the length of the data signal; and

byte alignment means for adapting data of differing lengths for placement into the shared storage area of the memory unit.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L12: Entry 1 of 22 File: USPT Oct 30, 2001

DOCUMENT-IDENTIFIER: US 6311204 B1
TITLE: Processing system with register-based process sharing

Abstract Text (1):
A method and apparatus for preventing interference between simultaneously-running processes in a set top box processing system which attempt to access certain shared processing hardware such as a drawing acceleration engine. A graphics processor or other device such as a CPU associated with the processor includes a register with an acquire bit portion and a process identifier portion. When a given process requests access to a graphics engine or other shared processing hardware, a determination is made as to whether the acquire bit of the register is set. A set acquire bit indicates that some process has already been granted access to the engine. If the acquire bit is not set, the requesting process is granted access to the engine, and its process identifier is stored in the process identifier portion of the register. If the acquire bit is already set when the given process requests access to the engine, the identifier for that process is compared to the identifier stored in the process identifier portion of the register. If the identifiers match, the requesting process is granted access. The lack of a match between the identifiers indicates that a different process has previously been granted access to the engine, and the requesting process is therefore denied access to the engine. When a process granted access to the engine no longer requires access, the acquire bit is cleared.

Application Filing Date (1):
19961011

Brief Summary Text (4):
Multimedia distribution systems are becoming increasingly important vehicles for delivering video, audio and other data to and from remote users. Such distribution systems include cable or community access television (CATV) systems, telephone systems and computer networks. A set top box may be used as an interface between the distribution system and a television set, computer or other type of remote user terminal. The set top box typically provides functions such as input/output processing of video, audio and other data, audio and video demultiplexing and decompression, graphics overlay processing for use in electronic program guides and the like, entitlement control for video on demand (VOD), near video on demand (NVOD) and pay-per-view (PPV) applications, and remote control user interfaces.

Brief Summary Text (14):
Another aspect of the invention involves a technique for utilizing a hardware register to prevent interference between simultaneously-running processes which attempt to access certain processing hardware such as a drawing acceleration engine. In a exemplary embodiment, a method and apparatus are provided for controlling access of a plurality of processes to a graphics engine in a graphics processor. The graphics processor or other device such as a CPU associated with the processor includes a register with an acquire bit portion and a process identifier portion. When a given process requests access to the graphics engine, a determination is made as to whether the acquire bit of the register is set. A set acquire bit indicates that some process has already been granted access to the

engine. If the acquire bit is not set, the requesting process is granted access to the engine, and its process identifier is stored in the process identifier portion of the register. If the acquire bit is already set when the given process requests access to the engine, the identifier for that process is compared to the identifier stored in the process identifier portion of the register. If the identifiers match, the requesting process is granted access. The lack of a match between the identifiers indicates that a different process has previously been granted access to the engine, and the requesting process is therefore denied access to the engine. When a process granted access to the engine no longer requires access, the acquire bit is cleared. This hardware-based sharing mechanism allows multiple processes to share common state-sensitive graphics hardware such as a drawing acceleration engine.

Brief Summary Text (15):
Another aspect of the invention is directed to a memory arbitration technique which allows multiple hardware functions implemented in a single ASIC to utilize a single shared memory unit or multiple shared memory units. The memory arbitration technique establishes a priority among multiple memory access requestors which is particularly well-suited for use in a set top box processing system. This aspect of the invention significantly reduces the complexity of a set top box or other processing system in that separate memory controllers are eliminated and memory conflicts are considerably reduced. An exemplary embodiment provides a method of arbitrating between a plurality of memory access requests received from a plurality of processing elements in a set top box processing system. The processing elements include a transport stream demultiplexer, a host central processing unit and a graphics processor. The method involves the steps of receiving the memory access requests from the processing elements, and permitting the processing elements to access a shared memory in accordance with an established priority. The established priority assigns a higher priority to the graphics processor than to the host central processing unit, and may be in the order of graphics processor, transport stream demultiplexer, and central processing unit. In an embodiment in which the plurality of processing elements includes an asynchronous transfer mode (ATM) processing element, the established priority may assign the lowest priority to the memory access requests of the ATM processing element.

Drawing Description Text (17):
FIG. 9A shows an exemplary register configured to provide a hardware-based drawing acceleration engine sharing function in accordance with the present invention.

Detailed Description Text (61):
The present invention provides clock circuitry which allows different elements in the set top box processing system 10 to operate with different but related system clocks. For example, the video data supplied from the MPEG-2 video decoder 52 to the ASIC processor 20 in the processing system 10 of FIG. 1 may be clocked by a first clock with a clock rate R1. The ASIC processor 20 may operate using a second clock with a clock rate R2, where R2 is a multiple of R1. The NTSC encoder 64 which receives the combined video/graphics output signal from the ASIC processor 20 may also operate with the first clock at rate R1. In one possible embodiment, the first clock may have a rate R1 of 27 MHz, while the second clock has a rate R2 of 1.5R1 or 40.5 MHz. Such non-integer variation in operating clock rates between different system elements has presented a number of problems in prior art processing systems, including metastability and difficulty in regulating pipelined data transfer. As a result, it has generally been necessary to utilize either a common or integer-related multiple clock for all elements of the processing system, or to provide complex regulation mechanisms designed to avoid metastability and to regulate data transfer. The present invention avoids these and other problems of prior art processing systems by utilizing a synchronous phase detector illustrated in FIG. 5A in conjunction with a multiplexed pipeline structure illustrated in FIG. 6A.

Detailed Description Text (106):

The graphics processor 60 in the ASIC processor 20 of FIG. 1 may implement a register-based sharing mechanism which prevents simultaneously-running processes from interfering in their attempts to access graphics acceleration engines or other portions of the graphics processor 60. The graphics processor 60 will generally include one or more hardware-based drawing acceleration engines as well as a graphics driver. An exemplary graphics driver suitable for use with the present invention is the MAUI driver available from Microware, Inc. of Des Moines, Iowa. The MAUI driver and many other graphics drivers will generally allow multiple applications to simultaneously generate and supply graphics data to the drawing acceleration engine. Each application may be configured as a user process, such that a task switch in the acceleration engine could be triggered at any time during a given process by a call directed to the acceleration engine. This may create a problem for the acceleration engine in that the color registers and other internal state information may become corrupted if two or more applications simultaneously attempt to use the acceleration engine. Possible solutions to this problem include making a kernel save and restore the state for any interrupted application, allowing only one application to use the acceleration engine, or using a software semaphore. However, each of these solutions may introduce additional complexities or other undesirable results. The present invention provides an approach based on a hardware semaphore which avoids many of these undesirable results, and will be described in greater detail below.

Detailed Description Text (107):
FIG. 9A shows an exemplary implementation of a semaphore register 350 used as a hardware semaphore in accordance with the invention. The semaphore register 350 includes an acquire bit portion 356, a process identifier- portion 354, and a remaining portion 356. In this example, the register 350 is configured as a 16-bit register, and may be contained within the graphics processor 60, the ASIC processor 20 or elsewhere in the processing system 10 of FIG. 1. The register 350 is used to control the access of multiple processes operating through a graphics driver to a drawing acceleration engine. The acquire bit portion 352 of the semaphore register 350 indicates to other processes that the drawing acceleration engine has been acquired by another process. The process which has acquired the drawing acceleration engine writes its identifier into the process identifier portion 354 of the register 350. Other processes can determine if the drawing acceleration engine has been acquired by simply examining the acquire bit portion of the register 350, and if the engine has not been acquired, may themselves acquire the engine.

Detailed Description Text (108):
FIG. 9B is a flow diagram illustrating the operation of the hardware semaphore feature of the present invention. In step 360, a given process operating through the graphics driver attempts to acquire the semaphore by writing its process identifier to the process identifier portion 354 of the semaphore register 350. Step 362 indicates that before the process is permitted to write to the register 350, a determination is made as to whether or not the acquired bit in the acquired bit portion 352 of the register 350 has been set. If the acquire bit has been set, step 364 indicates that the process is denied access to the drawing acceleration engine and the register 350 unless the process has the same process identifier as that already stored in the process identifier portion 354 of the register 350. If the acquire bit has not been set, the process attempting to acquire the semaphore stores its process identifier in the process identifier portion 354 of register 350, and is then permitted to utilize the drawing acceleration engine to the exclusion of other processes. In step 370, a determination is made as to whether the process has completed its use of the drawing acceleration engine. If the process has not completed its use of the engine, the process returns to step 368 and continues to use the engine. If the process has completed its use of the engine, the process clears the acquire bit in portion 352 of register 350, as shown in step 372. The cleared acquire bit indicates to other processes that the engine is now available to them. Although the hardware semaphore of the present invention

has been illustrated in conjunction with controlling the access of graphics processes to a drawing engine, it should be emphasized that this is by way of illustration and not limitation. The hardware semaphore may be utilized in other applications in which it is desirable to control the access of one or more processes to a state-sensitive device. These alternative applications include hardware acceleration circuitry for cyclic redundancy code (CRC) calculation, or any other type of shared processing resource.

Detailed Description Text (110):
The present invention provides memory arbitration techniques which allow multiple processes to share a common memory device or devices. In the exemplary processing system of FIG. 1, the memory arbitration techniques permit a number of graphics, communication and other processes operating within ASIC processor 20 to share the DRAM 40. This memory arbitration eliminates the requirement for separate memory devices in multiple processing elements, and thus permits a more efficient and cost-effective processing system implementation. Although illustrated below in conjunction with multiple system processes sharing a single memory device, it will be readily apparent that the disclosed techniques are also applicable to multiple processes sharing multiple memory devices.

Detailed Description Text (149):
FIG. 11A shows an exemplary SAR receiver 405 in accordance with the invention. The receiver 405 includes a receive state machine 420, receive logic 422 and a receive buffer 424. The receive, buffer 424 holds ATM cell data received from the UTOPIA port 400 until it can be processed in the receiver 405. The buffer 424 may be implemented as an 8.times.16 single-ported RAM in order to provide sufficient buffering for a 16-byte burst data transfer. The state machine 420 and logic 422 operate in conjunction with host CPU 30 and ASIC processor 20 to provide receive functions which are illustrated in the flow diagrams of FIGS. 11B and 11D below. The receiver 405 further includes a receive VCI look-up table 426 which may be implemented as a 16.times.16 RAM. The receive VCI table 426 contains information identifying the particular VCIs which are supported by the receiver 405. The receiver 405 accepts and processes a given incoming cell if that cell has a VCI which is found within the look-up table 426. The contents of the look-up table can be updated by software operating on host CPU 30. The table 426 may be stored in DRAM 40 or elsewhere within the processor 20. The receiver 405 may operate at a clock rate on the order of 20.25 MHz.

Detailed Description Text (163):
FIG. 13 illustrates an exemplary CRC processor 406 in greater detail. The processor 406 includes a CRC state machine 570, CRC logic 572 and a holding buffer 574. The CRC state machine 570 and CRC logic 572 combine to perform the above-noted CRC calculations in a well-known manner. The CRC processor 406 may operate at a clock rate of 40.5 MHz. The holding buffer 574 provides temporary storage of data blocks on which CRC calculations are to be performed, and may be implemented as an 8.times.16 single-ported RAM. The CRC processor 406 further includes a transmit accumulator register 576, a receive accumulator register 578, and a CRC command register 580. A CRC operation may be initiated on a- given data block by writing the start address of the block, the length of the block and a command into the command register 580. Exemplary commands which may be supported by the CRC processor 407 include commands requesting computation of a partial CRC for a receive or transmit cell or group of cells. Alternatively, a command may be provided for generating a cumulative CRC for any given set of receive or transmit cell data. The CRC calculation requested by the command written to register 580 is carried out in a conventional manner using the state machine 570 and logic 572. The results of the calculation are stored in the appropriate accumulator register 576 or 578. Separate accumulator registers are provided for receive and transmit in order to prevent interference between interruptable receive and transmit processes. The host CPU 30 interfaces with the registers 576, 578 and 580 of the CRC processor 406 to request and obtain the above-noted frame CRCs. In alternative embodiments,

the CRC function of ATM SAR 90 could be provided elsewhere in the ASIC processor 20 or in the CPU 30.

Detailed Description Text (165):
FIG. 14A is a block diagram of the SAR transmitter 407 incorporated into the ATM SAR 90. The SAR transmitter 407 includes a transmit state machine 602, transmit logic 604 and a transmit buffer 606. The transmit state machine 602 and transmit logic operate in conjunction with host CPU 30 and other portions of ASIC processor 20 to provide segmentation functions to be described in greater detail below. The transmit buffer 606 serves to buffer ATM cell data prior to its transmission via the UTOPIA port 400 and may be implemented as an 8.times.16 single-ported RAM. The SAR transmitter 407 may be configured to operate at a clock rate of 20.25 MHz.

CLAIMS:

1. A method of controlling access of a plurality of processes to a graphics engine in a graphics processor, the method including the steps of:

determining if an acquire indicator in a register has been set when a first process is attempting to access the graphics engine;

denying the first process access to the graphics engine if the acquire indicator has been set and a previously-stored process identifier does not match a process identifier of the first process; and

granting the first process access to the graphics engine if the acquire indicator is not set, or if the acquire indicator has been set and the previously-stored process identifier matches the process identifier of the first process, such that access to the graphics engine is granted based a first-come first-served paradigm.

2. The method of claim 1 wherein the graphics engine is a process-state sensitive drawing acceleration engine.

4. The method of claim 1 further including the step of storing a process identifier of the first process in a process identifier portion of the register if the first process is granted access to the graphics engine.

5. The method of claim 1 further including the step of setting the acquire indicator of the register if the first process is granted access to the graphics engine.

6. The method of claim 1 wherein the register further includes a process identifier portion for storing a process identifier for a process granted access to the graphics engine.

7. An apparatus for controlling access of a plurality of processes to a graphics engine in a graphics processor, the apparatus including:

a memory including a register for storing an acquire indicator indicating whether one of the plurality of processes has been granted access to the process; and

a processor coupled to the memory and operative to determine if the acquire indicator has been set when a first process is attempting to access the graphics engine, wherein the processor is further operative to deny the first process access to the graphics engine if the acquire indicator has been set and a previously-stored process identifier does not match a process identifier of the first process, and to grant the first process access to the graphics engine if the acquire indicator is not set or if the acquire indicator has been set and the previously-stored process identifier matches the process identifier of the first process, such that access to the graphics engine is granted based a first-come, first-served

paradigm.

8. The apparatus of claim 7 wherein the graphics engine is a process-state sensitive drawing acceleration engine.

10. The apparatus of claim 7 wherein the processor is further operative to store a process identifier of the first process in a process identifier portion of the register if the first process is granted access to the graphics engine.

11. The apparatus of claim 7 wherein the processor is further operative to set the acquire indicator of the register if the first process is granted access to the graphics engine.

12. The apparatus of claim 7 wherein the register further includes a process identifier portion for storing a process identifier for a process granted access to the graphics engine.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

L12: Entry 4 of 22 File: USPT Sep 14, 1999

DOCUMENT-IDENTIFIER: US 5953691 A
TITLE: Processing system with graphics data prescaling

Application Filing Date (1):
19961011

Brief Summary Text (4):
Multimedia distribution systems are becoming increasingly important vehicles for delivering video, audio and other data to and from remote users. Such distribution systems include cable or community access television (CATV) systems, telephone systems and computer networks. A set top box may be used as an interface between the distribution system and a television set, computer or other type of remote user terminal. The set top box typically provides functions such as input/output processing of video, audio and other data, audio and video demultiplexing and decompression, graphics overlay processing for use in electronic program guides and the like, entitlement control for video on demand (VOD), near video on demand (NVOD) and pay-per-view (PPV) applications, and remote control user interfaces.

Brief Summary Text (14):
Another aspect of the invention involves a technique for utilizing a hardware register to prevent interference between simultaneously-running processes which attempt to access certain processing hardware such as a drawing acceleration engine. In a exemplary embodiment, a method and apparatus are provided for controlling access of a plurality of processes to a graphics engine in a graphics processor. The graphics processor or other device such as a CPU associated with the processor includes a register with an acquire bit portion and a process identifier portion. When a given process requests access to the graphics engine, a determination is made as to whether the acquire bit of the register is set. A set acquire bit indicates that some process has already been granted access to the engine. If the acquire bit is not set, the requesting process is granted access to the engine, and its process identifier is stored in the process identifier portion of the register. If the acquire bit is already set when the given process requests access to the engine, the identifier for that process is compared to the identifier stored in the process identifier portion of the register. If the identifiers match, the requesting process is granted access. The lack of a match between the identifiers indicates that a different process has previously been granted access to the engine, and the requesting process is therefore denied access to the engine. When a process granted access to the engine no longer requires access, the acquire bit is cleared. This hardware-based sharing mechanism allows multiple processes to share common state-sensitive graphics hardware such as a drawing acceleration engine.

Brief Summary Text (15):
Another aspect of the invention is directed to a memory arbitration technique which allows multiple hardware functions implemented in a single ASIC to utilize a single shared memory unit or multiple shared memory units. The memory arbitration technique establishes a priority among multiple memory access requestors which is particularly well-suited for use in a set top box processing system. This aspect of the invention significantly reduces the complexity of a set top box or other

processing system in that separate memory controllers are eliminated and memory conflicts are considerably reduced. An exemplary embodiment provides a method of arbitrating between a plurality of memory access requests received from a plurality of processing elements in a set top box processing system. The processing elements include a transport stream demultiplexer, a host central processing unit and a graphics processor. The method involves the steps of receiving the memory access requests from the processing elements, and permitting the processing elements to access a shared memory in accordance with an established priority. The established priority assigns a higher priority to the graphics processor than to the host central processing unit, and may be in the order of graphics processor, transport stream demultiplexer, and central processing unit. In an embodiment in which the plurality of processing elements includes an asynchronous transfer mode (ATM) processing element, the established priority may assign the lowest priority to the memory access requests of the ATM processing element.

Drawing Description Text (17):
FIG. 9A shows an exemplary register configured to provide a hardware-based drawing acceleration engine sharing function in accordance with the present invention.

Detailed Description Text (61):
The present invention provides clock circuitry which allows different elements in the set top box processing system 10 to operate with different but related system clocks. For example, the video data supplied from the MPEG-2 video decoder 52 to the ASIC processor 20 in the processing system 10 of FIG. 1 may be clocked by a first clock with a clock rate R1. The ASIC processor 20 may operate using a second clock with a clock rate R2, where R2 is a multiple of R1. The NTSC encoder 64 which receives the combined video/graphics output signal from the ASIC processor 20 may also operate with the first clock at rate R1. In one possible embodiment, the first clock may have a rate R1 of 27 MHz, while the second clock has a rate R2 of 1.5R1 or 40.5 MHz. Such non-integer variation in operating clock rates between different system elements has presented a number of problems in prior art processing systems, including metastability and difficulty in regulating pipelined data transfer. As a result, it has generally been necessary to utilize either a common or integer-related multiple clock for all elements of the processing system, or to provide complex regulation mechanisms designed to avoid metastability and to regulate data transfer. The present invention avoids these and other problems of prior art processing systems by utilizing a synchronous phase detector illustrated in FIG. 5A in conjunction with a multiplexed pipeline structure illustrated in FIG. 6A.

Detailed Description Text (95):
The graphics processor 60 in the ASIC processor 20 of FIG. 1 may implement a register-based sharing mechanism which prevents simultaneously-running processes from interfering in their attempts to access graphics acceleration engines or other portions of the graphics processor 60. The graphics processor 60 will generally include one or more hardware-based drawing acceleration engines as well as a graphics driver. An exemplary graphics driver suitable for use with the present invention is the MAUI driver available from Microware, Inc. of Des Moines, Iowa. The MAUI driver and many other graphics drivers will generally allow multiple applications to simultaneously generate and supply graphics data to the drawing acceleration engine. Each application may be configured as a user process, such that a task switch in the acceleration engine could be triggered at any time during a given process by a call directed to the acceleration engine. This may create a problem for the acceleration engine in that the color registers and other internal state information may become corrupted if two or more applications simultaneously attempt to use the acceleration engine. Possible solutions to this problem include making a kernel save and restore the state for any interrupted application, allowing only one application to use the acceleration engine, or using a software semaphore. However, each of these solutions may introduce additional complexities or other undesirable results. The present invention provides an approach based on a hardware semaphore which avoids many of these undesirable results, and will be

described in greater detail below.

Detailed Description Text (96):
FIG. 9A shows an exemplary implementation of a semaphore register 350 used as a hardware semaphore in accordance with the invention. The semaphore register 350 includes an acquire bit portion 356, a process identifier portion 354, and a remaining portion 356. In this example, the register 350 is configured as a 16-bit register, and may be contained within the graphics processor 60, the ASIC processor 20 or elsewhere in the processing system 10 of FIG. 1. The register 350 is used to control the access of multiple processes operating through a graphics driver to a drawing acceleration engine. The acquire bit portion 352 of the semaphore register 350 indicates to other processes that the drawing acceleration engine has been acquired by another process. The process which has acquired the drawing acceleration engine writes its identifier into the process identifier portion 354 of the register 350. Other processes can determine if the drawing acceleration engine has been acquired by simply examining the acquire bit portion of the register 350, and if the engine has not been acquired, may themselves acquire the engine.

Detailed Description Text (97):
FIG. 9B is a flow diagram illustrating the operation of the hardware semaphore feature of the present invention. In step 360, a given process operating through the graphics driver attempts to acquire the semaphore by writing its process identifier to the process identifier portion 354 of the semaphore register 350. Step 362 indicates that before the process is permitted to write to the register 350, a determination is made as to whether or not the acquired bit in the acquired bit portion 352 of the register 350 has been set. If the acquire bit has been set, step 364 indicates that the process is denied access to the drawing acceleration engine and the register 350 unless the process has the same process identifier as that already stored in the process identifier portion 354 of the register 350. If the acquire bit has not been set, the process attempting to acquire the semaphore stores its process identifier in the process identifier portion 354 of register 350, and is then permitted to utilize the drawing acceleration engine to the exclusion of other processes. In step 370, a determination is made as to whether the process has completed its use of the drawing acceleration engine. If the process has not completed its use of the engine, the process returns to step 368 and continues to use the engine. If the process has completed its use of the engine, the process clears the acquire bit in portion 352 of register 350, as shown in step 372. The cleared acquire bit indicates to other processes that the engine is now available to them. Although the hardware semaphore of the present invention has been illustrated in conjunction with controlling the access of graphics processes to a drawing engine, it should be emphasized that this is by way of illustration and not limitation. The hardware semaphore may be utilized in other applications in which it is desirable to control the access of one or more processes to a state-sensitive device. These alternative applications include hardware acceleration circuitry for cyclic redundancy code (CRC) calculation, or any other type of shared processing resource.

Detailed Description Text (99):
The present invention provides memory arbitration techniques which allow multiple processes to share a common memory device or devices. In the exemplary processing system of FIG. 1, the memory arbitration techniques permit a number of graphics, communication and other processes operating within ASIC processor 20 to share the DRAM 40. This memory arbitration eliminates the requirement for separate memory devices in multiple processing elements, and thus permits a more efficient and cost-effective processing system implementation. Although illustrated below in conjunction with multiple system processes sharing a single memory device, it will be readily apparent that the disclosed techniques are also applicable to multiple processes sharing multiple memory devices.

Detailed Description Text (138):
FIG. 11A shows an exemplary SAR receiver 405 in accordance with the invention. The receiver 405 includes a receive state machine 420, receive logic 422 and a receive buffer 424. The receive buffer 424 holds ATM cell data received from the UTOPIA port 400 until it can be processed in the receiver 405. The buffer 424 may be implemented as an 8.times.16 single-ported RAM in order to provide sufficient buffering for a 16-byte burst data transfer. The state machine 420 and logic 422 operate in conjunction with host CPU 30 and ASIC processor 20 to provide receive functions which are illustrated in the flow diagrams of FIGS. 11B and 11D below. The receiver 405 further includes a receive VCI look-up table 426 which may be implemented as a 16.times.16 RAM. The receive VCI table 426 contains information identifying the particular VCIs which are supported by the receiver 405. The receiver 405 accepts and processes a given incoming cell if that cell has a VCI which is found within the look-up table 426. The contents of the look-up table can be updated by software operating on host CPU 30. The table 426 may be stored in DRAM 40 or elsewhere within the processor 20. The receiver 405 may operate at a clock rate on the order of 20.25 MHz.

Detailed Description Text (152):
FIG. 13 illustrates an exemplary CRC processor 406 in greater detail. The processor 406 includes a CRC state machine 570, CRC logic 572 and a holding buffer 574. The CRC state machine 570 and CRC logic 572 combine to perform the above-noted CRC calculations in a well-known manner. The CRC processor 406 may operate at a clock rate of 40.5 MHz. The holding buffer 574 provides temporary storage of data blocks on which CRC calculations are to be performed, and may be implemented as an 8.times.16 single-ported RAM. The CRC processor 406 further includes a transmit accumulator register 576, a receive accumulator register 578, and a CRC command register 580. A CRC operation may be initiated on a given data block by writing the start address of the block, the length of the block and a command into the command register 580. Exemplary commands which may be supported by the CRC processor 407 include commands requesting computation of a partial CRC for a receive or transmit cell or group of cells. Alternatively, a command may be provided for generating a cumulative CRC for any given set of receive or transmit cell data. The CRC calculation requested by the command written to register 580 is carried out in a conventional manner using the state machine 570 and logic 572. The results of the calculation are stored in the appropriate accumulator register 576 or 578. Separate accumulator registers are provided for receive and transmit in order to prevent interference between interruptable receive and transmit processes. The host CPU 30 interfaces with the registers 576, 578 and 580 of the CRC processor 406 to request and obtain the above-noted frame CRCs. In alternative embodiments, the CRC function of ATM SAR 90 could be provided elsewhere in the ASIC processor 20 or in the CPU 30.

Detailed Description Text (154):
FIG. 14A is a block diagram of the SAR transmitter 407 incorporated into the ATM SAR 90. The SAR transmitter 407 includes a transmit state machine 602, transmit logic 604 and a transmit buffer 606. The transmit state machine 602 and transmit logic operate in conjunction with host CPU 30 and other portions of ASIC processor 20 to provide segmentation functions to be described in greater detail below. The transmit buffer 606 serves to buffer ATM cell data prior to its transmission via the UTOPIA port 400 and may be implemented as an 8.times.16 single-ported RAM. The SAR transmitter 407 may be configured to operate at a clock rate of 20.25 MHz.

Previous Doc Next Doc Go to Doc#

L12: Entry 16 of 22 File: USPT Mar 15, 1994

DOCUMENT-IDENTIFIER: US 5295258 A
TITLE: Fault-tolerant computer system with online recovery and reintegration of redundant components

Abstract Text (1):
A computer system in a fault-tolerant configuration employs multiple identical CPUs executing the same instruction stream, with multiple, identical memory modules in the address space of the CPUs storing duplicates of the same data. The system detects faults in the CPUs and memory modules, and places a faulty unit offline while continuing to operate using the good units. The faulty unit can be replaced and reintegrated into the system without shutdown. The multiple CPUs are loosely synchronized, as by detecting events such as memory references and stalling any CPU ahead of others until all execute the function simultaneously; interrupts can be synchronized by ensuring that all CPUs implement the interrupt at the same point in their instruction stream. Memory references via the separate CPU-to-memory busses are voted at the three separate ports of each of the memory modules. I/O functions are implemented using two identical I/O busses, each of which is separately coupled to only one of the memory modules. A number of I/O processors are coupled to both I/O busses. I/O devices are accessed through a pair of identical (redundant) I/O processors, but only one is designated to actively control a given device; in case of failure of one I/O processor, however, an I/O device can be accessed by the other one without system shutdown.

Application Filing Date (1):
19900105

Detailed Description Text (5):
Each one of the memory modules 14 and 15 is separately coupled to a respective input/output bus 24 or 25, and each of these busses is coupled to two (or more) input/output processors 26 and 27. The system can have multiple I/O processors as needed to accommodate the I/O devices needed for the particular system configuration. Each one of the input/output processors 26 and 27 is connected to a bus 28, and each bus 28 is connected to one or more bus interface modules 29 for interface with a standard I/O controller 30 which may be of the VMEbus.TM. type. Each bus interface module 29 is connected to two of the busses 28, so failure of one I/O processor 26 or 27, or failure of one of the bus channels 28, can be tolerated. The I/O processors 26 and 27 can be addressed by the CPUs 11, 12 and 13 through the memory modules 14 and 15, and can signal an interrupt to the CPUs via the memory modules. Disk drives, terminals with CRT screens and keyboards, and network adapters, are typical peripheral devices operated by the controllers 30. The controllers 30 may make DMA-type references to the memory modules 14 and 15 to transfer blocks of data. Each one of the I/O processors 26, 27, etc., has certain individual lines directly connected to each one of the memory modules for bus request, bus grant, etc.; these point-to-point connections are called "radials" and are included in a group of radial lines 31.

Detailed Description Text (213):
Local memory 16 is restored by using the DMA engine 74 to copy each block of local memory 16 out to global memory 14, 15, and back again; this copy-back has the

effect of copying good memory to the bad. This technique relies upon two features of the system construction; first, the contents of local memory 16 are preserved across a soft reset of the CPU, and, second, the DMA engine 74 always runs to completion-in the case of a vote error, the consensus of the data will be used, and at the end of the transfer status will indicate which CPU failed the vote.

Other Reference Publication (11):
Hopkins, Jr., "A Fault-Tolerant Information Processing Concept for Space Vehicles", IEEE Trans. on Computers, Nov. 1971, pp. 1394-1403.

Other Reference Publication (18):
S. Chang, "Multiple-Read Single Write Memory and its Applications", IEEE Transactions on Computers, Aug. 1990, pp. 689-694.

Other Reference Publication (20):
Malaiya, Y., "Fault-Tolerance in Multiple Processor Systems", Proc. IEEE Int'l. Conf. on Circuits and Computers, Oct. 1-3, 1980, Port Chester, N.Y., pp. 710-716.

Other Reference Publication (21):
Wensley, J., "Fault-tolerant computers ensure reliable industrial controls", Electronic Design, Jun. 25, 1981, pp. 129-135.

Other Reference Publication (23):
Wensley, J., "Industrial-control system does things in threes for safety", Electronics, Jan. 27, 1983, pp. 98-102.

CLAIMS:

1. A method of operating a computer system having multiple CPUs executing the same instruction stream, the CPUs each having local memory and also each accessing multiple global memory units storing identical data, comprising the steps of:

a) detecting an error in one of said CPUs;

b) isolating said one CPU from the system and continuing to execute said instruction stream and accessing said global memory units by the other ones of said CPUs;

c) reintegrating said one CPU after rendering said CPU operative by first bringing said one CPU into sync with said other ones of said CPUs by soft-resetting all of said multiple CPUs prior to continuing normal operation of said multiple CPUs, said soft-resetting non-destructively preserving the current state and the local memory of each said multiple CPU, then restoring the state and the local memory of said one CPU to be identical to the state and the local memory of the said other ones of the CPUs.

2. A method according to claim 1 wherein any one of the global memory units may be designated as primary for the purpose of supplying read data to said multiple CPUs and the others of the said global memory units are designated backup.

3. A method according to claim 1 wherein said step of restoring the state and the local memory includes;

a) copying each state variable of the other ones of the CPUs to global memory and then copying each state variable from global memory to the appropriate state register in all of said multiple CPUs;

b) copying a portion of local memory of the other ones of the CPUs to global memory and then copying said portion from global memory to local memory in all of said multiple CPUs;

c) repeating step b) for different portions of local memory of the other ones of the CPUs until all variables stored in local memory of the other ones of the CPUs have been copied to global memory and then copied from global memory to all of said multiple CPUs.

9. A method according to claim 6 wherein said step of restoring the state and the memory contents of global memory includes:

a) configuring said one global memory unit to ignore all access requests from I/O Processors;

b) reading each global memory unit processor board state variable from the primary global memory unit to said multiple CPUs and storing said processor board state variable from the multiple CPUs to all global memory units including said one global memory unit;

c) reading each local memory data word stored in the primary global memory unit to said multiple CPUs and storing said local memory data word from the multiple CPUs to all global memory units including said one global memory unit;

d) repeating step c;

e) configuring said one global memory unit to execute all access requests from I/O Processors.

Previous Doc Next Doc Go to Doc#

L12: Entry 19 of 22 File: USPT Oct 23, 1990

DOCUMENT-IDENTIFIER: US 4965718 A
TITLE: Data processing system incorporating a memory resident directive for synchronizing multiple tasks among plurality of processing elements by monitoring alternation of semaphore data

Application Filing Date (1):
19880929

Brief Summary Text (6):
In a true parallel processing system, each of the multiple processors has access to shared common memory, has access to at least a portion of the system input/output (I/O), and is controlled by a single operating system providing interaction between the processors and the programs they are executing. Theoretically, then, it is possible to divide a large program between N parallel tasks, each task running in a separate processor, and complete the program a factor of N times faster than any single processor could complete the job alone.

Brief Summary Text (7):
Many different system configurations are known for connecting the multiple processors, and related system memory and I/O elements, to function in the manner described above. These configurations include time-share bus configurations wherein the system elements are interconnected via a time-shared data link, crossbar configurations wherein the system elements are connected via an arrangement of matrix switches, and multiple-bus/multiport systems wherein processing and I/O elements are connected to multiple memory ports via multiple buses. Each system configuration has associated with it different advantages and disadvantages, many of which are still under investigation and open to debate between those skilled in the art. For a general discussion of multiprocessor performance, the reader is directed to an article in the IEEE PROCEEDINGS OF THE 1985 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, pgs. 772-781, "A Methodology for Predicting Multiprocessor Performance", by A. Norton, et al. For a more thorough description of one particular type of parallel processing system, the reader is directed to an article in the IEEE PROCEEDINGS OF THE 1985 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, pages 764-771, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture", by G.F. Pfister, et al. References to the IBM RP3 parallel processor will be made throughout this document for the purpose of exemplifying features typically found in parallel processing systems. It is to be understood that the invention set out below is in no way limited by the constructs of the RP3 system.

Brief Summary Text (13):
One already known method of diminishing the undesirable formation of these hot spots is that of combining multiple fetch or read requests for a single memory location. According to this method, the responsibility for notifying all of the processors waiting on the particular memory location is assigned to a single processor. This method, while functioning to some extent to relieve hot spots, is subject to several disadvantages. First, the efficiency of such combination is dependant on the lucky collisions or overlapping of requests for the same memory location. Such schemes require additional code and storage resources to manipulate

the lists. The cost in hardware of building the interconnect networks required to support such combining is very high. Further, if the single processor having the notification responsibility should fail, continued operation of the system may be seriously spots in general, the reader is directed to an article in the IEEE PROCEEDINGS OF THE 1986 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, pgs. 28-34, "The Onset of Hot Spot Contention", by M. Kumar, et al. For a discussion of hot spots and combining, the reader is directed to an article in the IEEE PROCEEDINGS OF THE 1985 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, pgs. 790-797, "`Hot Spot` Contention and Combining in Multistage Interconnection Networks", by G.F. Pfister, et al.

Brief Summary Text (22):
The principal object of the present invention is to provide a new and improved method and apparatus for communicating data between multiple tasks in data processing systems.

Brief Summary Text (24):
A further object of the present invention is to provide such a method and apparatus for communicating data between multiple tasks in a uniprocessor processing system.

Brief Summary Text (25):
Yet another object of the present invention is to provide such a method and apparatus for communicating data between multiple tasks in a multiple processing system.

Brief Summary Text (26):
A more specific object of the present invention is to provide a method and apparatus for communicating data between multiple tasks in a processing system through the use of a directive operative in a memory element to monitor the status of a semaphore.

Detailed Description Text (4):
Referring now to FIG. 2, the features of one exemplary processing element 12-N are shown in greater detail. Processing element 12-N includes a computing engine 18, preferably including an arithmetic logic unit (ALU) and a floating point processor (FPP), conventional features well known to those skilled in the art. A memory mapping controller 20 is connected to computing engine 18 for mapping the memory addresses generated by the computing engine to memory elements 14 (FIG. 1) including a local memory space 46A designated for the exclusive use of each processing element 12 (but not necessarily located with the processing element hardware). An interface 22 is connected to memory mapping controller 20 for providing a logical interface with interconnection network 16. A controller 24 is associated with each processing element 12 for controlling the operation of the various components therein. It will be understood that, while the interconnections between the various elements may be indicated with single lines, the drawings are intended to show logical connections, and the actual signal lines may comprise buses of multiple conductors supporting multiple, parallel signals.

Detailed Description Text (5):
In the operation of processing element 12, control logic 24 functions to control computing engine 18 in accordance with directives stored in memory including local memory 46A. Memory mapping controller 20 functions to convert memory addresses generated by computing engine 18 to accommodate various memory paging and interleaving schemes as discussed in further detail below. Interface 22 provides a logical interface for exchanging digital data between memory mapping controller 20 and interconnection network 16.

Detailed Description Text (14):
While the memory elements 14 have been shown as M discrete elements each supporting DRAM memory 46, it will be understood that in an actual implementation the memory

can comprise one or more elements, and is typically paged and interleaved in accordance with many such schemes well known to those skilled in the art. Memory mapping controller 20 of processing element 12 functions to convert the address generated by computing engine 18 in accordance with the selected paging and/or interleaving scheme. For a more thorough discussion of memory addressing in parallel processing systems, the reader is directed to "Computer Architecture and Parallel Processing", by Hwang and Briggs, McGraw-Hill, Inc. 1984, pgs. 60-118. A detailed description of the operation of memory elements 14 in parallel processing system 10 is set out hereinbelow with reference to FIGS. 6-8.

Detailed Description Text (16):
Referring again back to FIG. 1, the type of interconnection network 16 utilized in system 10 is not relevant to the understanding or practice of the present invention. Referring to the articles on "hot spots" referenced above, the reader will see that the problem addressed by the present invention arises in parallel processing systems regardless of the type of interconnection network used. Whether interconnection network 16 is selected to be a bus network, a matrix switching network, a multi-bus/multiport network, or some combination of the above, it must include a tree structure for interconnecting multiple sinks (i.e. processors, I/O devices, etc.) to every shared memory location. Hence, regardless of the type of interconnection network used, the present invention can be utilized to minimize the occurrence of hot spots resulting from memory access contentions. As an example of one particular type of interconnection network, the reader is directed to an article describing that used in the IBM RP3 parallel processor discussed above: IEEE PROCEEDINGS OF THE 1985 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, pages 764-771, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture", by G.F. Pfister, et al.

Detailed Description Text (18):
FIG. 5 shows a flow chart of a process including a parent task indicated at 5A, and N child tasks indicated at 5B, these parent and child tasks having been parsed by an operating system for parallel processing in multiple processing elements. The parent task 5A initializes the data required by the N child tasks 5B for parallel processing, and then signals the child tasks with a count of "-1" (a semaphore) stored in a shared memory location indicated as FLAG1. The parent task then enters a spin loop, indicated in dashed-line at 60, wherein it repeatedly fetches the count stored in FLAG1 to determine if all of the children have completed their processing, i.e. if the stored count equals the number N of child tasks. The parent task will remain in this spin loop, repeatedly fetching the count stored at the FLAG1 memory location until the value read indicates the child tasks are all completed. After the fetched value indicates that the child tasks are completed, the parent task will collect the results of the child tasks and store a "-1" count in a separate FLAG2 memory location to indicate to the child tasks that they are all complete, and the data is collected.

Detailed Description Text (24):
Once the CAN directive is assembled, it is transmitted to memory element 14 (step 602) via an appropriate electronic "handshake" with control unit 52. To provide this electronic handshake, the requesting processing element 12 checks the clear to send CTS signal line to be sure the selected memory element 14 is free to receive a directive, and then sends an appropriate REQUEST TYPE signal identifying a CAN directive along with a VALID IN signal and the assembled CAN directive.

Detailed Description Text (43):
In accordance with yet another embodiment of the present invention, the CAN directives are beneficially utilized in a uniprocessor processing system. Such a uniprocessor system is substantially identical to that described above with respect to FIGS. 1, 2, 3, and 4, above, with the exception that only a single processing element 12 and memory element 14 are provided, the pair being connected by an appropriate bus-type interconnection network 16. The operation of such a

uniprocessor system is likewise substantially identical to that shown and described with respect to FIGS. 6 and 7 above, excepting that each CAN directive must include a task identification, the various tasks being executed serially on the single processing element 12 (vs. in parallel on multiple processing elements as shown in FIGS. 7A, 7B).

Detailed Description Text (45):
There is thus provided a new and improved method and apparatus for communicating data between multiple tasks in processing systems through the use of memory directives operative to monitor the status of a semaphore at a memory location provided by a requesting processor. When utilized in a multiprocessor/parallel processing system, the present invention provides for the communication of data between a plurality of processing elements, through semaphores stored in a plurality of memory elements, without repetitious accessing of those memory elements over an interconnection network. The present invention thus substantially reduces hot spots otherwise formed within the interconnection network, increasing the speed and efficiency of the parallel processing system. Fine grain parallelism is thus accommodated in large scale parallel processing systems. When utilized in a uniprocessor environment, the present invention provides for the communication of data between multiple tasks without high overhead polling by the processing element. The processing element is thus freed to more efficiently process data.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L19: Entry 1 of 8 File: USPT Oct 10, 2000

DOCUMENT-IDENTIFIER: US 6131155 A
TITLE: Programmer-visible uncached load/store unit having burst capability

Application Filing Date (1):
19980123

Brief Summary Text (11):
In addition, it is unlikely that the CPU will make more than one access to each of a small number of words of the packet during the modification process. Thus, if the data cache were to read the packet from main memory in the normal manner, there would be considerable wastage of memory bandwidth, as a substantial number of words would be read but only a small number of words would be actually modified by the CPU for each packet being processed. As the Ethernet packet switch is expected to process thousands of such packets per second, with consecutive packets being stored in widely different memory locations, the net loss of efficiency is considerable. A solution that has been used in some specialized CPUs is to dispense with the data cache altogether, and use a queued approach to accessing memory. (This is also referred to as a decoupled access/execute architecture.) In such a system, the CPU is permitted to notify a memory access control unit (that regulates accesses to the main memory) in advance that an access will be made to specific data words. The use of queues within the memory access control unit permits multiple such notifications to be made by the CPU, well in advance of when the data are actually required from the main memory during program execution. The memory access control unit is then responsible for sorting out the requests for memory data, capturing such locality as may exist, and fetching (or storing) the data from the main memory in the most efficient manner possible. The memory access control unit also uses a set of queues to return the data to the CPU, which may then, at some future time, read the data out of these queues. The memory access execute unit, in conjunction with the CPU program, thus eliminates the latency effects incurred by random accesses to the main memory from impacting the program execution. At the same time, it avoids the memory consistency problem by not maintaining data copies indefinitely. In effect, the decoupled access/execute architecture renders the memory access mechanism visible to the programmer of the CPU (who has to cause the advance notifications of memory accesses to be generated by inserting appropriate instructions into the program).

Previous Doc Next Doc Go to Doc#

L19: Entry 2 of 8 File: USPT Feb 3, 1998

DOCUMENT-IDENTIFIER: US RE35723 E
TITLE: Synchronous burst-access memory

Application Filing Date (1):
19951204

Brief Summary Text (11):
Yet another object of the invention is to facilitate the use of memory in multiple-processor systems.

Detailed Description Text (88):
The advantage of the seventh novel memory is that a processor can delegate memory access to a memory management unit and execute other operations without having to count accessed bits. At the completion of memory access, the flag signal notifies the processor (or other external devices) that the memory is available to begin the next burst access.

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L19: Entry 4 of 8 File: USPT Apr 2, 1996

DOCUMENT-IDENTIFIER: US 5504670 A
TITLE: Method and apparatus for allocating resources in a multiprocessor system

Abstract Text (1):
A computer system having multiple processors and multiple resources for use by the processors when executing their assigned tasks. The computer system includes a plurality of sub-controllers which allow the resources within the computer system to be allocated among the processors, such that each of the processors has the resources required to complete its task.

Application Filing Date (1):
19930331

Brief Summary Text (13):
An integrated circuit tester for use in testing multiple integrated circuits is described. The tester includes multiple processing elements. Each of the processing elements executes a task designed to test one type of integrated circuit, such that a multiple tasks are executed by the processors. The tester includes multiple resources which are used by the processing elements to complete their tasks. The tester also includes means to allocate the resources to each of the processing elements, wherein each processing elements has the resources that are necessary to perform its task, such that different types of integrated circuits may be tested at the same time.

Detailed Description Text (10):
Note that in such an arrangement, the sub-controller of the present invention may also use the resources of its associated processor. For instance, if a sub-controller wishes to access the system memory in its resources grouping, then the sub-controller notifies its processor that a read or write to memory must be performed and then obtains control over common bus 410. In order to accomplish this, the processor is placed in a hold state so that the sub-controller retains control of the resources on the bus. In the currently preferred embodiment, the sub-controller places its processor in a hold state using a hold signal. Referring back to FIG. 4, hold signals 406B-409B are shown coupled from sub-controllers 406A-409B respectively to processors 402-405 respectively. In this configuration, each sub-controller is able to be the master, while its associated processor is a slave. Thus, the present invention provides a sub-controller which acts as both a master and a slave.

Previous Doc Next Doc Go to Doc#

L19: Entry 5 of 8 File: USPT Jun 6, 1995

DOCUMENT-IDENTIFIER: US 5423008 A
TITLE: Apparatus and method for detecting the activities of a plurality of processors on a shared bus

Application Filing Date (1):
19920803

Brief Summary Text (7):
In addition to a multiplicity of specialized processors, computer system architects increasingly incorporate multiple general-purpose processors in their designs. This helps escape processing limitations of a single general-purpose processor. One way additional general-purpose processors are included is through a design's provision for intelligent adapter cards. Intelligent adapter cards may contain both general-purpose adapter processors and specialized processors that collect or manipulate data before passing it to a general-purpose processor.

Brief Summary Text (8):
Alternately, computer system architects may include multiple general-purpose processors that collectively process information. Finally, computer system architects may include both multiple general-purpose processors and intelligent adapter cards in a design.

Brief Summary Text (9):
In any of these approaches, architects necessarily include provisions for synchronizing the various processors. Specifically, architects must pay particular attention to the mechanisms a plurality of processors use to synchronize and communicate activities. As an example, multiple processors that share a common data memory must detect when other processors change data values stored in the memory. This allows processors to invalidate their caches, causing a subsequent cache refresh for the data item's changed value when it is needed. Cache updates occur at a hardware level and are transparent to a processor's software. Such hardware cache-coherency considerations are well documented in John L. Hennessy's and David A. Patterson's Computer Architecture A Quantitative Approach published by Morgan Kaufmann Publishers Inc., 1990.

Detailed Description Text (8):
Referring now to FIG. 2, a preferred embodiment of the bus signal detection mechanism 26 of the present invention is shown. In the preferred embodiment, the bus signal detection mechanism 26 and processor A 2 are components on the same integrated circuit. Processor A 2 couples the shared system bus 6 and the private bus A 14. Using the shared system bus 6, processor A 2 can access the address space of shared memory 8. The bus signal detection mechanism 26 notifies processor A 2 when processor B 4 accesses specified memory addresses in the address space of shared memory 8. The bus signal detection mechanism 26 preferably comprises a plurality of event registers 50, 52, an event masking component 68, a comparator 78, a processor access detector 96, and a processor ID filter 104

Detailed Description Text (18):
To prevent unwanted event notifications to processor A 2 that might occur when

processor A 2 itself accesses the shared memory 8, the processor access detector 96 couples to a local processor signal 94 from processor A 2 and shared system bus 6. This allows the processor access detector 96 to generate an inhibit signal 98 sent to the comparator 78. The bus signal detection mechanism 26 ignores access events that occur while the inhibit signal 98 is TRUE. Processor A 2 could optionally disable the processor access detector 96 by a variety of well known methods. This would allow multi-processing program threads under processor A's control to generate events that other program threads under processor A's control respond to. Also, if the shared system bus 6 provides bus processor ID signals 100 that indicate which processor 2, 4 is generating the access on the shared system bus 6, that signal 100 could feed the processor access detector 96 directly, precluding the need for other local bus signals such as the local processor signal 94.

Previous Doc Next Doc Go to Doc#

L19: Entry 6 of 8 File: USPT Jul 5, 1994

DOCUMENT-IDENTIFIER: US 5327390 A
**** See image for Certificate of Correction ****
TITLE: Synchronous burst-access memory

Application Filing Date (1):
19930915

Brief Summary Text (11):
Yet another object of the invention is to facilitate the use of memory in multiple-processor systems.

Detailed Description Text (87):
The advantage of the seventh novel memory is that a processor can delegate memory access to a memory management unit and execute other operations without having to count accessed bits. At the completion of memory access, the flag signal notifies the processor (or other external devices) that the memory is available to begin the next burst access.

L25: Entry 1 of 29 File: USPT Oct 15, 2002

DOCUMENT-IDENTIFIER: US 6467012 B1
**** See image for Certificate of Correction ****
TITLE: Method and apparatus using a distributed system structure to support bus-based cache-coherence protocols for symmetric multiprocessors

Application Filing Date (1):
19990708

Detailed Description Text (21):
FIG. 4 shows nodes 410 and 420, which contain groupings of system elements. The number of nodes may vary based on the configuration of the system. Node 410, also labeled as Node.sub.0, contains processors 411 and 412, also labeled as Processor P.sub.0 and Processor P.sub.P-1, which are the masters for Node 410. Each node controller has multiple standard bidirectional processor address-data buses over which masters are connected into the distributed system. Processors 411 and 412 connect to node controller 415, also labeled as Node Controller NC.sub.0, via buses 413 and 414, also labeled as P.sub.0 Bus and P.sub.p-1 Bus, respectively. Node 420, also labeled as Node.sub.N-1, contains processor 421 and I/O agent 422, which are the masters for Node 420. Processor 421 and I/O device 422 connect to node controller 425, also labeled as Node Controller NC.sub.N-1 via buses 423 and 424, respectively. The number of masters per node may vary depending upon the configuration of the system, and the number of masters at each node is not required to be uniform across all of the nodes in the system.

Detailed Description Text (85):
In the distributed SMP system described in FIGS. 4-10D, multiple processors and other devices can issue transactions simultaneously over the multiple buses in the system. Thus, at the outset, there is ambiguity regarding the order of the transactions as they are issued. As they flow through the system, as a first step, the system imposes a "heuristic order of arrival" over them that is reasonable and fair. This preliminary order is not necessarily the order in which the transactions eventually complete in the system. If two colliding transactions are simultaneously active in the system, the one that ranked "earlier of the two" by the heuristic order of arrival will be slated to be completed first if coherence does not require otherwise.

Current US Cross Reference Classification (1):
709/213

CLAIMS:

4. The data processing system of claim 1 wherein each node controller comprises: a plurality of master device ports, wherein each master device port connects to a master device bus; a pair of address switch ports, each one of said pair of address switch ports connecting to one of a pair of unidirectional address switch buses, said pair of unidirectional address switch buses also being connected to said address switch, wherein one of the pair of address switch buses conveys an address from the node controller to the address switch and the other one of the pair of address switch buses conveys an address from the address switch to the node

controller; and a plurality of memory subsystem ports, each one of said plurality of memory subsystem ports connecting to a different, single bidirectional memory subsystem bus, said bidirectional memory subsystem bus also being connected to a different one of said plurality of memory subsystems, wherein a memory subsystem bus conveys data between the node controller and one of the memory subsystems.

6. The data processing system of claim 5 wherein said data controller comprises: a plurality of master device data ports, wherein each master device data port connects to a data portion of a master device bus; and a plurality of memory subsystem ports, each one of said plurality of memory subsystem ports connecting to a different, single bidirectional memory subsystem bus, said bidirectional memory subsystem bus also being connected to a different one of said plurality of memory subsystems, wherein a memory subsystem bus conveys data between the data controller and the memory subsystem that is connected to the memory subsystem bus.

Previous Doc Next Doc Go to Doc#

L25: Entry 5 of 29 File: USPT Jun 26, 2001

DOCUMENT-IDENTIFIER: US 6253290 B1
TITLE: Multiprocessor system capable of circumventing write monitoring of cache memories

Application Filing Date (1):
19990210

Brief Summary Text (22):
U.S. Pat. No. 4,939,641 discloses a method that possesses shared/unshared information in the cache memory, and carries out read and write of the cache using the write back method for the unshared data, and the write through method for the shared data. In summary, they employs a method with "write monitoring". There are countless such configurations comprising multiple processors and cache memories, and some of them assume the "write monitoring".

Detailed Description Text (21):
The present invention implements a high speed and low cost system by circumventing the write monitoring by placing the contents that are used only in an instant processing, that is, the contents of a work area, only within the local cache memories, and by writing the contents that are used by the plurality of processor units into the single memory without placing them into the local cache. Here, let us take an example of obtaining average marks of five subjects by five CPUs.

Detailed Description Text (148):
The CPU main unit 386 executes, by a single memory access instruction on software, not only a read operation of the segment descriptor table but also an access operation of the segment. When accessing the segment descriptor table 136, the CPU main unit 386 outputs a memory access request in accordance with a predetermined access procedure, and produces the "ReadSGT" signal from the Sgr terminal 131d. The shared/unshared bus selectors 122, 123 and 124 are disabled owing to the "ReadSGT" supplied to their enable terminals connected to the Sgr terminal 131d, and hence all the buses are disconnected.

Current US Cross Reference Classification (1):
709/213

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L25: Entry 19 of 29 File: USPT Mar 16, 1999

DOCUMENT-IDENTIFIER: US 5884055 A
TITLE: Method and apparatus including a shared resource and multiple processors running a common control program accessing the shared resource

Application Filing Date (1):
19961127

Brief Summary Text (12):
Each Director includes a Motorola 68030 microprocessor to control its onboard pipelines moving data between the host Channel and Global Memory (if it is a CA), or moving data between the Global Memory and the disk array (if it is a DA). The 68030 microprocessor also requires access to Global Memory, which it can not directly address. Two pipes are provided on each Director to provide facilities for the 68030 to access Global Memory. A Direct Single Access (DSA) pipe facilitates transfers of a single memory word at a time and is typically used for control/status type operations. A Direct Multiple Access (DMA) pipe can transfer from 1 to 8 memory words on each memory access and is thus more efficient for transferring larger blocks of data. The DMA pipe is typically used by the 68030 to transfer large amounts of data for testing/setting up Global Memory.

Brief Summary Text (18):
Each Global Memory board provides for two port access by the Directors via the two independent buses designated "A" and "B". Global Memory supports a burst transfer feature for all of the pipes whereby 1 to 8 memory words can be transferred sequentially to/from memory for all of the pipes in a single memory access, except for the DSA pipe which only transfers a single memory word at a time. The memory on each board is organized in banks of 8 to facilitate the burst mode transfers. The system operates most efficiently with a burst size of 8 and with a starting address aligned on an 8-word boundary, that is, when the starting address starts at bank 0 and continues up to include bank 7. Each word transfer is clocked by a burst clock signal generated on the memory board, however, the Director is responsible for calculating a final address for each burst transfer. At the end of each transfer, the Director makes a check between address bits and upper pointer bits, which have been incremented by the burst clock, to ensure integrity of the transfer.

Brief Summary Text (19):
As previously indicated, there is independent arbitration for the Global Memory. There are three possible users for the memory array on a Global Memory board: the "A" port or bus; the "B" port or bus; and refresh. Refresh has the highest priority while the A and B ports operate on a rotating priority. The A and B buses effectively combine on the memory board and therefore only one bus can be in use on any single memory board at any time. A Director will request a port to the A or B bus by driving the address and command lines, and asserting a select line for the port under control of the 68030 and control program. A port will be available for use: if it is not in refresh; if any other accesses to it are complete; if the port is not locked through the other port, or if the other port is not in use. If a port is successfully selected, it will return a bus grant signal to the Director selecting it, which will remain asserted until the transfer is complete.

Brief Summary Text (25):
Integrated cache according to the invention, i.e. Global Memory, comprises a plurality of memory boards, with variations of 256 Mbyte, 512 Mbyte, or 1 Gbyte of DRAM cache memory. A system can access up to 32 Gbytes of cache. In order to take advantage of the increased processor speeds of the dual control processors according to the invention, the memory boards support burst transfers of up to 32 memory words. The increased burst size, i.e. amount of data transferred between a Dual Port Ram (DPR) and Global Memory, allows for the transfer of 1 to 32 memory words, 64 bits in length, in a single memory access.

Detailed Description Text (4):
Global Memory in a system according to the invention comprises a plurality of memory boards, four in the present illustrative embodiment, that provide from 256 Mbyte to 4 Gbyte of Dynamic Random Access Memory (DRAM) for cache memory. Each board uses up to 576 16 Mbit DRAM memory devices arranged on both sides of four daughter cards. The global memory boards provide for two port access from one or more controllers or adapters configured either to transfer data between a host and Global Memory (front end adapters), or to transfer data between the Global Memory and the disk array (back end adapters). The Global Memory is accessed by the front end or back end adapters, referred to generically as Directors, via two independent buses, an even bus and an odd bus, and two respective ports designated "A" and "B". The A and B ports each consist of a 32-bit address bus plus 1 parity bit, a 64-bit data bus plus 8 ECC bits, 10 command lines plus 1 parity bit, and bus control lines. The term memory word is used herein to denote a 64 bit plus 8 ECC bit memory word. The memory boards support burst transfers of 64 bit memory words through the A and B ports. The burst transfer feature allows for the sequential transfer of 1 to 32 memory words in a single memory access.

Detailed Description Text (6):
The memory boards have two modes of operation. In a first mode or "operational mode", normal read/write transfers to the Global Memory are effected, as described in detail hereinafter. Operational mode is effectively an on-line/off-line flag signalling the Directors that the Global Memory may be selected for data transfers, including: single memory word write/read via a Directors DSA pipeline; burst read/write of 1 to 32 memory words via an ESCON, Channel or SCSI pipe (depending upon the type of Director). In a second mode or "non-operational mode" a board is non-operational or off-line, and only service cycle reads or writes are possible. Service cycle reads or writes do not write to the Global Memory array and are used for initial set-up and checking memory board status. After power up memory boards wake up in the non-operational state and must be initialized before being available for normal operational memory cycles. During initialization service cycles are used in the non-operational mode to load the segment tables that define what ranges of addresses the board will respond to. Service cycles are also used to read and write status and control information to Maintenance Processors and Memory Controllers on the memory boards.

Detailed Description Text (7):
Service cycles may be used in certain instances when a memory board is in operational mode or on-line. A write service cycle may be used when a memory board is on-line, to write single word to memory board ASICs. Similarly, a read service cycle may be used in operational mode for a single word read from memory board ASICs. During service cycles, in either operational or non-operational modes, memory boards are selected by means of their slot number whereas during normal operational cycles they are accessed by a global memory address.

Detailed Description Text (16):
The Memory Controller on each memory board also handles bus arbitration. There are three possible users of the memory array on a global memory board: the A port; the B port; and refresh. Only one of these users may be active at any time and which one is determined by onboard (Memory Controller) arbitration circuitry. Arbitration

is handled in a substantially similar manner to that of the Symmetrix ICDA. Refresh has the highest priority while the A and B ports operate on a rotating priority. The A and B buses effectively combine on the memory board and therefore only one bus can be in use on any single memory board at any time. A given port will be available for use: if it is not in refresh; if any other accesses to it are complete; if the port is not locked through the other port, or if the other port is not in use.

Detailed Description Text (29):
As described, Global Memory constituted by individual memory boards configured as described hereinbefore, is written to and read from over the system bus or backplane by controllers referred to generically herein as "Directors", constituted by controllers/adapters transferring data between the Global Memory and the disk array (back end). A plurality of lines or signals comprise the (backend) Director/Global Memory interface. The following signals represent the signals between a Director and a single memory port, i.e. A or B, and are effectively duplicated for a second port.

Detailed Description Text (55):
The two microprocessors on each of the Directors described herein also each require access to Global Memory, which, from the microprocessors point of view is not directly addressable. Each processor has two pipes with which to access Global Memory. A DSA (Direct Single Access) pipe which transfers a single 64-bit memory word at a time, is typically used for control/status type operations. A DMA (Direct Multiple Access) pipe can transfer from 1 to 32 64-bit memory words on each memory access and is thus more efficient for transferring larger blocks of data. The DMA pipe is generally used to transfer large amounts of data for testing/setting up Global Memory.

Detailed Description Text (89):
Since the DSA pipe only transfers a single memory word per transfer there is no need to increment the pointer values. In the case of the upper pointer for the DSA pipe, the "next" pointer is still written back to the ADGA by the machine, the difference with the DSA pipe being that the "next" pointer should be the same as the previous pointer as no incrementing takes place. The current pointer value can be read from the ADGA by the controlling X or Y processor at any time. The lower pointer is written/read on bits (15:0) of the data bus while the upper pointer is on bits (31:16). This allows the processor to read or write both pointers in one bus cycle and thus makes for more efficient operation.

Current US Cross Reference Classification (1):
709/213